UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

DYNAMICAL QUALITY OF SERVICE OVER SOFTWARE DEFINED NETWORKING

A Degree's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Héctor Pelicano Gómez

In partial fulfilment

of the requirements for the degree in

TELEMATIC ENGINEERING

Advisor: Luís J. de la Cruz Llopis

Barcelona, January 2015

Abstract

The purpose of this project is to demonstrate that SDN networks can offer a dynamic quality of service and develop software which can offer practical examples.

SDN concept refers to a new model of architecture network which allows uncoupling the application of network infrastructure and the services provided, allowing network administrators to define all parameters and functions independently of the provider. Here is where the network administrator can define the established quality policies and manage them according to the necessities.

Nowadays, these networks are adopting important relevance and it is expected in a near future they will be able to replace the present networks.

Resum

El propòsit d'aquest projecte és demostrar que es pot oferir una qualitat de servei dinàmica en xarxes definides per software, anomenades xarxes SDN, i desenvolupar un software que permeti oferir exemples pràctics.

El terme SDN fa referència a un nou model d'arquitectura de xarxa. Permet desacoblar la infraestructura de la xarxa de les aplicacions i serveis que aquesta proporciona. Això permet als administradors de xarxa definir completament tots els paràmetres i comportaments de la xarxa sense dependre del proveïdor. És aquí on l'administrador de la xarxa pot definir completament les polítiques de qualitat establertes i gestionar-les en tot moment en funció de les seves necessitats.

Avui en dia, aquestes xarxes estan adquirint moltíssima importància i s'espera que en un futur no massa llunyà puguin arribar a substituir les xarxes actuals.

Resumen

El propósito de este proyecto es demostrar que se puede ofrecer calidad de servicio dinámica sobre redes definidas por software, llamadas redes SDN, y desarrollar un software que permita ofrecer ejemplos prácticos.

El término SDN hace referencia a un nuevo modelo de arquitectura de red. Permite desacoplar la infraestructura de la red de las aplicaciones y servicios que ésta proporciona. Esto permite a los administradores de red definir completamente todos los parámetros y comportamientos de la red sin depender del proveedor. Es aquí donde el administrador de la red puede definir completamente las políticas de calidad establecidas y gestionarlas en todo momento en función de las necesidades.

Hoy en día, estas redes están adquiriendo muchísima importancia y se espera que en un futuro no muy lejano puedan llegar a substituir las redes actuales.

Dedicado a mis padres por aguantarme tantos años y a mi novia por su apoyo y ayuda.

También a todos los kapos de Telecogresca, por haber hecho de mis años en la universidad el mejor periodo de mi vida.

Agradecimientos

Me gustaría dar mi más sincero agradecimiento al tutor de mi proyecto: Luis J. de la Cruz. Primero de todo, por haberme enseñado en sus clases a amar las redes telemáticas y por darme la oportunidad de investigar sobre un tema tan actual y tan interesante como es SDN. En segundo lugar, por prestarme oído y ayuda siempre que lo he necesitado.

Por otro lado, quisiera también dar las gracias a todos los desarrolladores de Floodlight por facilitar el trabajo y, en especial, a Ryan Wallner por su módulo de QoS.

Finalmente, me gustaría dar las gracias a todos los profesores que he tenido durante la carrera, ya que no sólo me han enseñado conceptos, sino que también, han contribuido a formarme como persona.

Historial de revisiones y registro de aprobación

Revisión	Fecha	Propósito
0	14/01/2015	Creación del documento
1	20/01/2015	Revisión del documento
2	26/01/2015	Revisión del documento
3	02/02/2015	Aprobación del documento

LISTA DE DISTRIBUCIÓN

Nombre	e-mail
Héctor Pelicano Gómez	peli.tlc@gmail.com
Luís J. de la Cruz Llopis	luis.delacruz@entel.upc.edu

Escrito por:		Revisado y aprobado por:	
Fecha	14/01/2015	Fecha	02/02/2015
Nombre	Héctor Pelicano Gómez	Nombre	Luís J. de la Cruz Llopis
Puesto	Autor del proyecto	Puesto	Supervisor del proyecto

Tabla de contenidos

Listado de Figuras

1. Introducción

El objetivo general de este proyecto es estudiar y comprobar mediante virtualización el comportamiento de las redes SDN para la provisión de diferentes calidades de servicio a distintos flujos de tráfico en función de sus necesidades. Se tratará también la posibilidad de adaptación dinámica de la calidad de servicio ofrecida.

Los objetivos más específicos son los mencionados a continuación:

- Familiarización con los entornos de trabajo SDN.
- Identificación de los parámetros QoS que se tratarán.
- Aprendizaje del controlador Floodlight que permitirá ofrecer QoS dinámica.
- Creación del programa que permita controlar redes SDN con QoS dinámica.

1.1. SDN (Software Defined Networking)

Desde los inicios de internet (década de los años 80) las redes de comunicación han crecido siguiendo los estándares de organizaciones como el IEEE o el ICANN. Estas han utilizado los múltiples protocolos que permiten la interconexión entre redes de diferentes fabricantes o tecnologías.

Este sistema que ha funcionado hasta la actualidad, ha empezado a quedarse obsoleto dado el rápido crecimiento de internet y a la rigidez que las redes tradicionales presentan. Un claro ejemplo de esta rigidez se pudo observar el 3 febrero del año 2011. En esa fecha, las direcciones IP versión 4 se agotaron oficialmente y se necesitó el uso de nuevos mecanismos como la NAT o la creación de las direcciones IP versión 6.

Como respuesta a estas limitaciones de las redes clásicas ha surgido un nuevo modelo de redes definidas por software. Estas redes son conocidas como redes SDN.

1.1.1. Definición.

El término SDN hace referencia a un nuevo modelo de arquitectura de red que permite desacoplar la infraestructura de la red de las aplicaciones y servicios que ésta proporciona. Esto permite a los administradores de red definir completamente todos los parámetros y comportamientos de la red sin depender del proveedor.

SDN simplifica también los dispositivos de red puesto que ya no necesitan entender y procesar miles de protocolos, sino simplemente aceptar instrucciones de los controladores SDN. Estos controladores utilizan el protocolo OpenFlow para transmitir las ordenes del plano de control al plano de datos y, así, poder dar ordenes constantemente a la red para adecuar su tipología a cada necesidad.

1.1.2. Arquitectura.

Tradicionalmente las redes de comunicación han consistido en un conjunto de elementos de conmutación unidos por medios de transmisión. Estas redes tienen una gestión distribuida, donde cada elemento de conmutación incorpora un firmware que toma decisiones en función de unos protocolos marcados a priori.

Como se puede ver en la *Figura 1*, las redes SDN son físicamente iguales que la tradicionales y se diferencian de estas mediante la incorporación de una capa de control sobre la capa física, de manera que la gestión de la red se realiza mediante software por un controlador externo. Se puede apreciar también, la separación en SDN del plano de datos con el plano de control.

Otro cambio significativo que se puede apreciar es el tipo de switch usado, ya que el switch SDN es un conmutador mucho más simple y más económico que el tradicional.

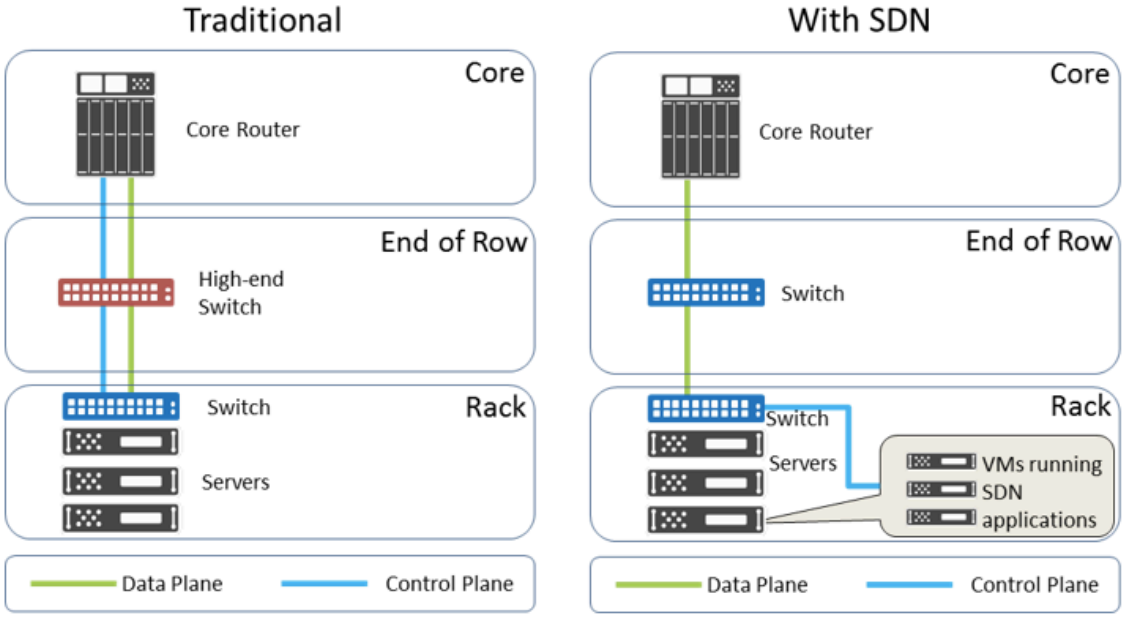

Figura 1: *Comparativa de la arquitectura de una red tradicional y otra SDN*

1.1.3. Controladores.

Los controladores son los dispositivos principales en la arquitectura SDN. Son estos quienes toman decisiones, implementan las reglas de la red, ejecutan las instrucciones proporcionadas por las aplicaciones y las distribuyen a los elementos de conmutación de la capa física. Se puede decir que los controladores son el auténtico cerebro de las redes SDN y que los elementos de la capa física tan solo aceptan ordenes de estos.

Actualmente, se pueden encontrar muchos tipos de controladores; algunos varían en función del lenguaje en el que están basados o la funcionalidad final para la que han sido creados. De esta manera podemos encontrar tanto controladores SDN especializados en gestión de servicios web como especializados en el control de redes escalables

1.2. QoS (Calidad de Servicio)

Cada día miles de personas acceden por primera vez a internet y pasan a formar parte de la gran red de redes que es hoy en día la World Wide Web. Estos usuarios (aproximadamente unos 3.000 millones) generan y a la vez descargan una inmensa cantidad de datos. Todos estos contenidos son solicitados por usuarios que tienen redes con características muy diferentes pero que esperan recibir la información con una cierta calidad.

Desde los inicios de internet la velocidad de transmisión ha sido el principal parámetro que ha interesado a los usuarios, ya que la mayoría de contenidos eran estáticos y no importaba demasiado el retardo. Sin embargo, actualmente, muchas aplicaciones de videollamadas, VoIP, videos bajo demanda y juegos online entre otras, han acentuado la necesidad de que los datos lleguen con el mínimo retardo y la mayor velocidad posible.

Una alternativa para suplir estas necesidades sería proporcionar una conexión de alta calidad que ofrezca una red con un exceso de capacidad, de modo que sea capaz de asumir los picos de tráfico sin congestiones ni pérdidas. Estas redes con sobre-aprovisionamiento, tienen el principal inconveniente de no ser escalables, ya que a cada usuario que se una a una red troncal, la capacidad de esta red ha de poder asumir el caudal total del usuario y esto se traduce en infraestructuras muy costosas.

Otra alternativa que nos ofrecen algunas tecnologías de conmutación es la QoS que utilizando mecanismos de control, permiten aprovechar de una manera más inteligente los recursos existentes.

1.2.1. Definición.

La *calidad de servicio* es definida por la Unión Internacional de Telecomunicaciones (UIT) como el efecto global de la calidad de funcionamiento de un servicio que determina el grado de satisfacción de un usuario de dicho servicio.

Esta definición deja en función del cliente cuáles son las características y comportamientos que lo satisfacen (minimizar el retardo, asegurar velocidad mínima, priorizar tráficos, etc.) ya que cada uno puede tener unas necesidades diferentes. Es por ello que se plantea el problema de poder ofrecer una calidad de servicio dinámica, que pueda moldearse en todo momento a los requisitos del usuario y que no se ofrezca como un servicio rígido.

La calidad de servicio cobra importancia cuando la capacidad de la red es insuficiente, especialmente para aplicaciones de streaming multimedia: voz sobre IP, juegos en línea y IP-TV en tiempo real. Estos a menudo requieren velocidad de bits fija y son sensibles al retardo. Un ejemplo donde la capacidad es un recurso limitado sería la comunicación de datos celular.

1.2.2. Soportes de la QoS.

Actualmente la calidad de servicio no se ofrece en todos los sistemas de conexión a la red, ya que muchos de los soportes más usados como Ethernet o Frame Relay no admiten niveles de QoS.

Un soporte que sí que admite QoS es ATM (Asynchronous Transfer Mode-ATM) donde un proveedor de ATM puede ofrecer tanto retardo mínimo como un ancho de banda específico para cierto servicio. La tecnología ATM permite un mejor aprovechamiento de las canales troceando la información proveniente de diferentes fuentes, creando así pequeños paquetes que son agrupados en el denominado *Módulo ATM*, para ser transportados por grandes enlaces de transmisión.

1.2.3. Mecanismos de control.

En el campo de redes de telecomunicaciones de conmutación de paquetes se pueden encontrar mecanismos de control para la reserva de recursos. Esta reserva de recursos se basa en proporcionar prioridades a diferentes aplicaciones, usuarios o flujos de datos, o para garantizar un cierto nivel de rendimiento a un flujo de datos. Por ejemplo, pueden ser garantizadas una tasa de bits, un retardo, una fluctuación, la probabilidad de pérdida de paquetes requerida o la tasa de error de bit priorizando un tipo de tráfico sobre el resto.

Quienes pueden garantizar estas situaciones son los mecanismos de control. Se pueden encontrar gran variedad de mecanismos basados, por ejemplo, en el control por ventana o el control por tasa. Otro tipo de control interesante puede ser el de asignación de ciertos recursos a un host en concreto, o a los paquetes de cierta aplicación.

2. Estado de la tecnología utilizada en esta tesis:

A fin de lograr combinar las redes definidas por software y la calidad de servicio, se han buscado herramientas para poder facilitar la tarea. Los tres elementos básicos para un entorno de trabajo son un controlador SDN, un simulador de redes y un módulo de QoS.

Como resultado de la búsqueda, se ha decidido usar el entorno de simulación de redes virtuales Mininet y el controlador SDN Floodlight que incluye de serie un módulo de QoS. Se han barajado otras opciones de controladores más usados, y pese a que Floodlight se encuentra en fase beta y genera muchos errores, se ha apostado por él. Los principales motivos son que dispone de una documentación muy completa y útil, está programado en java y sobretodo porqué tiene un módulo de QoS incorporado.

Otro de los controladores que se analizó fue Opennass, que si bien aparentaba ser más estable y sencillo de usar que Floodlight, no disponía de mucha documentación ni ejemplos.

2.1. Floodlight

Floodlight es un controlador de redes SDN, de código abierto y basado en java. Admite switches tanto físicos como virtuales que entiendan el lenguaje Openflow a la vez que admite redes mixtas donde existan algunos switches convencionales.

Floodlight incorpora también una serie de aplicaciones REST de control de redes que permite la configuración y obtención de datos de redes mediante comandos GET y POST. Además permite añadir otras aplicaciones externas de manera modular. Entre estas aplicaciones modulares ya creadas por otros programadores y compatibles con Floodlight, se encuentra un módulo de QoS y otro de Firewall. Más adelante se verá que estos módulos serán fundamentales para lograr el objetivo de este proyecto.

Finalmente Floodlight ofrece un interfaz web donde se pueden visualizar los componentes de nuestra red, las conexiones y el resto de parámetros que la definen.

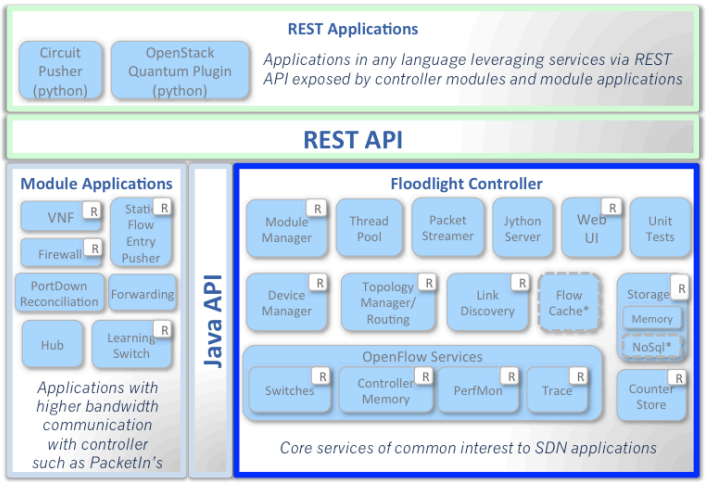

Figura 2: *Esquema del controlador Floodlight*

2.2. Mininet

Mininet es un simulador de redes que funciona sobre Linux y que permite crear hosts, enlaces, controladores y switches que entienden el lenguaje Openflow. Arrancado desde una máquina virtual, permite crear una red SDN completa en segundos, además de permitir el acceso de un controlador SDN externo en el momento de la creación. Las topologías de circuitos se pueden hacer a medida, tan complejas como se desee, a través de scripts Python.

Mininet permite acceder a cada host individualmente y ejecutar las aplicaciones instaladas en el sistema desde estos. De esta manera, se pueden monitorizar las conexiones de cada host mediante la herramienta Wireshark y también se pueden crear servidores webs dentro de cada host.

Todo esto hace que Mininet sea el simulador de redes SDN por excelencia. Dado que es un proyecto Open Source, está en constante evolución, permitiendo además que cualquier usuario pueda modificar el código con total libertad.

3. Desarrollo del proyecto:

3.1. Preparación del entorno

Este proyecto ha sido desarrollado sobre un sistema OS X 10.9.5 y se han necesitado una serie de elementos para preparar el entorno de trabajo:

- Apache Ant v1.9.4. Que nos permitirá automatizar la compilación del controlador SDN.

- Java for OS X 2014-001. Que instala los complementos necesarios para poder programar con Java.

- Xcode. Entorno de desarrollo de OS X donde se incluyen funcionalidades y repositorios de java.

- Xquartz. Programa que soporta las aplicaciones X11. Nos permitirá abrir aplicaciones y servicios de una máquina virtual en remoto.

- Eclipse. Plataforma sobre la que se trabajará para editar y compilar el código Java del controlador Floodlight.

- Virtualbox. Programa que permite crear máquinas virtuales a partir de imágenes de disco.

(Estos programas y herramientas se pueden obtener gratuitamente en internet o en la App Store de Apple.)

3.1.1. Preparación de Mininet

En primer lugar se procederá a descargar una imagen de Mininet de la web oficial y que preferiblemente sea la versión 2.0.0.

Una vez se tengan instalados todos los programas definidos previamente y la imagen de Mininet descargada, se procederá a la creación de la máquina virtual que permitirá recrear una red SDN. Usando VirtualBox se cargará la imagen descargada y se configurará la máquina de la siguiente manera:

- Utilizar como mínimo 1 GB de memoria RAM, ya que se necesitará para poder trabajar con tipologías complejas y altas cargas de trabajo. Para este proyecto se han utilizado 2 GB.

- En la configuración de redes se necesitarán 2 adaptadores. Para ello se tendrán que configurar previamente las redes disponibles que ofrece el programa VirtualBox. En la configuración del programa, y en el apartado de redes se añadirá una del tipo “solo-anfitrión” llamada ‘*vboxnet0’.* Una vez listo el programa, se pasará a la configuración de la máquina virtual. Como se puede ver en las *Figuras 3* y *4*, en el apartado de redes se conectará el primer adaptador a la red NAT y, el segundo, a la red “solo-anfitrión” (host-only) creada previamente. Esta configuración permitirá el acceso a internet mediante la red NAT y la comunicación con la máquina host sobre la que se trabaja mediante la red “solo-anfitrión”.

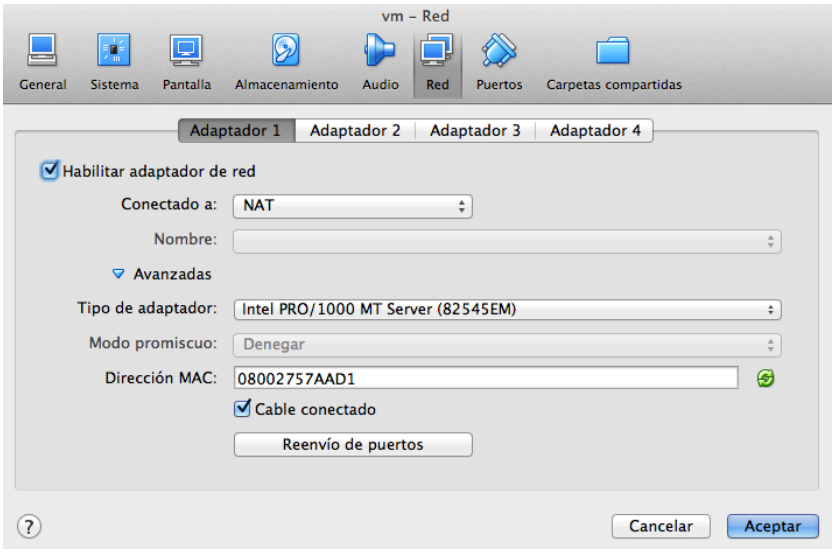

Figura 3: *Configuración del adaptador 1*

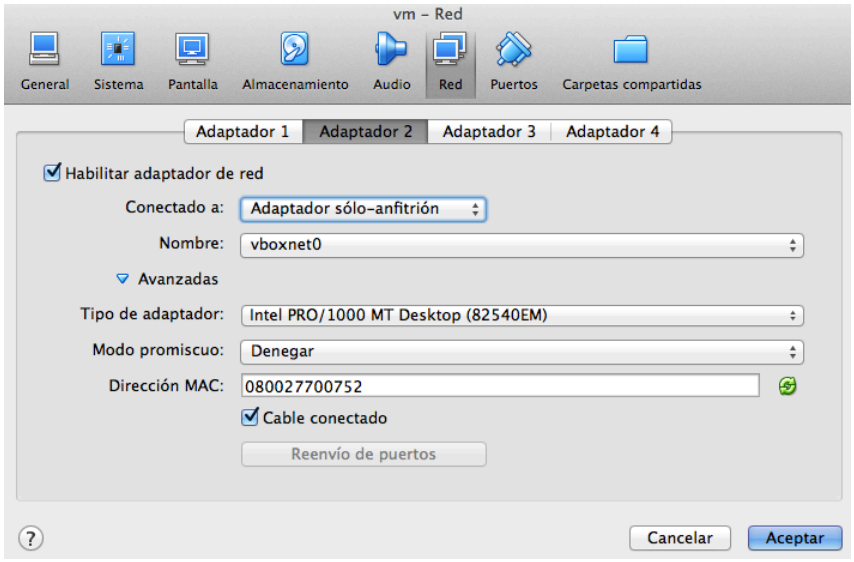

Figura 4: *Configuración del adaptador 2*

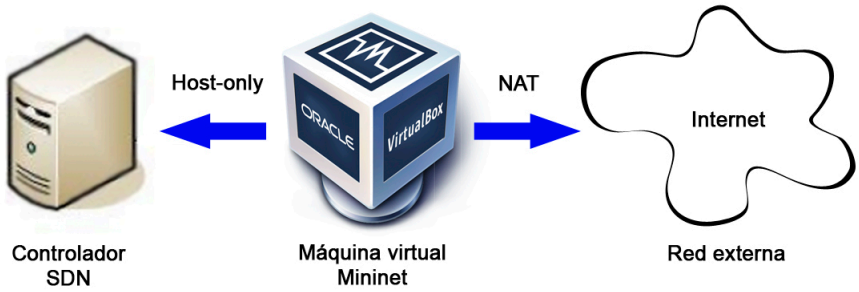

Figura 5: *Conexiones de la máquina virtual*

Con los adaptadores de red y la memoria configuradas, se procederá a iniciar la máquina virtual. Tras el arranque la máquina solicitará hacer login. Las máquinas Mininet descargadas de la web oficial, tanto el nombre de usuario como la contraseña son "mininet".

Una vez dentro de la máquina se verificará si se dispone de la versión 1.4.3 del repositorio Ovs-vsctl, el cual permitirá añadir diferentes colas a los switches virtuales. Algunas versiones de Mininet lo traen incorporado de serie, pudiéndolo verificar ejecutando:

$ sudo ovs-vsctl -V

En el caso de no tener la versión 1.4.3 o de no tenerlo instalado, se necesitará la instalación del repositorio. El tutorial se puede encontrar en la web oficial de los desarrolladores openvswitch.com.

El servicio de Ovs-vsctl requiere ser iniciado por primera vez mediante el comando:

$ /etc/init.d/openvswitch-switch start

Una vez se tenga instalado y ejecutado el servicio Ovs-vsctl se podrán hacer las primeras pruebas con Mininet para verificar su correcto funcionamiento.

Siempre que se inicie la máquina, el primer paso será el de mirar si los interfaces de red están correctamente configurados. Para ello se ejecutará el comando:

$ ifconfig

Con este comando se verán los interfaces eth0, eth1 y lo entre otros. Para poder trabajar correctamente con ellos, necesitaremos que eth0 y eth1 dispongan de direcciones IP. Generalmente eth1 no nos aparecerá configurado (Figura 6):

```
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:57:aa:d1
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe57:aad1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:49 errors:0 dropped:0 overruns:0 frame:0
          TX packets:66 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6364 (6.3 KB)  TX bytes:5760 (5.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:176 errors:0 dropped:0 overruns:0 frame:0
          TX packets:176 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:14032 (14.0 KB)  TX bytes:14032 (14.0 KB)
```

Figura 6: *Configuración de red donde no aparece eth1*

Para configurar correctamente el interfaz eth1 se ejecutará:

$ sudo dhclient eth1

Tras ejecutar este comando se puede verificar la correcta configuración (*Figura 7*) de todos los interfaces volviendo a ejecutar:

$ ifconfig

```
mininet@mininet-vm:~$ sudo dhclient eth1
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:57:aa:d1
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe57:aad1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:62 errors:0 dropped:0 overruns:0 frame:0
          TX packets:99 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7254 (7.2 KB)  TX bytes:8174 (8.1 KB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:70:07:52
          inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe70:752/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1438 (1.4 KB)  TX bytes:922 (922.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:176 errors:0 dropped:0 overruns:0 frame:0
          TX packets:176 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:14032 (14.0 KB)  TX bytes:14032 (14.0 KB)
```

Figura 7: *Configuración de red con eth1 configurado correctamente*

En la imagen de Mininet usada en este proyecto, este proceso fue necesario cada vez que se iniciaba la máquina.

Para facilitar el manejo y poder tener más de una consola a la vez abierta en Mininet, se recomienda realizar conexiones ssh desde la máquina host. Para realizarla tan solo hay que abrir un terminal y realizar la conexión:

$ ssh -X mininet@192.168.56.101

En este momento la máquina ya es completamente funcional, pudiendo verificar el correcto funcionamiento creando una red Mininet básica:

$ sudo mn

```
mininet@mininet-vm:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
```

Figura 8: *Red básica mininet*

Como se puede ver en la *Figura 9*, la red está formada por dos hosts y un switch controlado por la máquina Mininet. A esta topología se le llama *mínima*.

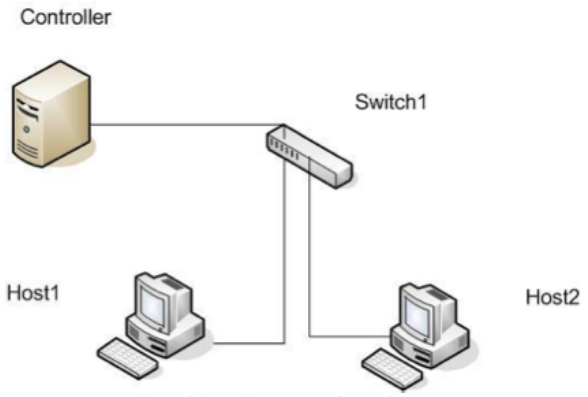

Figura 9: *Topología mínima*

Dado que el controlador pasa a ser la propia máquina virtual, el prompt pasará a ser "**mininet>**" y, por lo tanto, se podrá controlar la red. Algunos de los comandos que podemos ejecutar desde el controlador son:

- **mininet> net** // muestra información sobre la red
- **mininet> h1 ping –c1 h2** // manda un ping desde el Host 1 (h1) al Host 2 (h2)
- **mininet> h1 ifconfig** // muestra información sobre los interfaces de h1
- **mininet> exit** // cierra el controlador

Como se puede ver en estos comandos, Mininet permite realizar acciones desde cada uno de los hosts como si fuera el propio equipo. De esta manera, no solo se pueden realizar acciones básicas (ping, ifconfig,…) sino que también, permite ejecutar todo tipo de programados instalados en estos Hosts.

Una vez se cierre el controlador Mininet se recomienda limpiar la información dejada por la red previa ejecutando:

$ sudo mn –c

El comando "mn" nos permite añadir diversas características a nuestra red, generar todo tipo de tipologías y ceder el control a un controlador externo.

Para el ejemplo que se presentará más adelante, necesitaremos una tipología lineal compuesta por: 4 switches y 4 hosts, donde los switches permitan la adición de diferentes canales a cada puerto y que el controlador SDN sea el ordenador físico sobre el que está trabajando.

Lo primero que se necesitará será la dirección IP de la máquina host sobre la que se trabaja (***$ifconfig*** en OSX; ***$ipconfig*** en Windows). Una vez se tenga, se añadirá en el siguiente comando:

$ sudo mn -x --topo linear,4 --switch ovsk --controller=remote,ip=X.X.X.X:6633 --ipbase 10.0.0.0/8

Este comando generará una tipología de 4 switches conectados entre si, además de un host conectado a cada switch. El controlador se asigna en remoto, y la dirección IP será la del ordenador host sobre el que estamos trabajando, de manera que cuando se inicie el controlador Floodlight, este se conectará a la red creada. Se puede verificar la tipología de la red ejecutando:

mininet> net

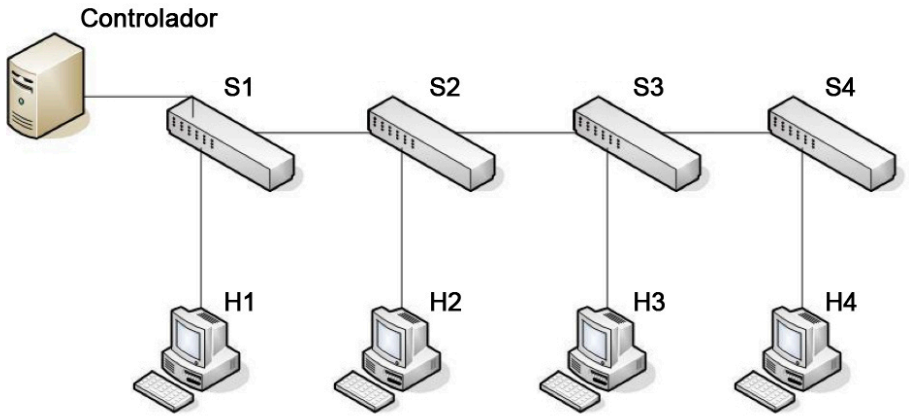

Figura 10: *Tipología de red*

```
mininet> net
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3 s3-eth3:s4-eth2
s4 lo:  s4-eth1:h4-eth0 s4-eth2:s3-eth3
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
```

Figura 11: *Detalle de las conexiones*

Finalmente, después de crear la red, se añadirán canales de diferentes capacidades a cada puerto de cada switch. Este proceso se puede hacer manualmente ejecutando la configuración deseada para switch, pero el comando es muy extenso y complicado de recordar. Para agilizar el proceso se puede ejecutar un script Python incluido en el paquete “floodlight-qos-beta” llamado “mininet-add-queues.py”.

Por defecto este script crea 3 canales de diferentes capacidades (1Gb puerto 0, 20Mb puerto 1 y 2Mb puerto 2) en cada puerto, aunque como se verá más adelante, se puede modificar en función de cada necesidad.

Para conseguir el script, se procederá a copiar el paquete Floodlight-qos-beta desde Github al directorio deseado dentro de la máquina virtual Mininet. Si previamente se ha creado la red virtual, el prompt de la máquina virtual será "**mininet>**", por lo tanto se procederá a crear una nueva conexión ssh desde la máquina host para poder seguir trabajando sin necesidad de cerrar el controlador. Una vez se tenga iniciada una nueva sesión ssh se ejecutará:

$git clone https://github.com/wallnerryan/floodlight-qos-beta.git

Una vez se haya copiado el proyecto y, con la red creada previamente, se configurarán los puertos ejecutando:

$sudo ./floodlight-qos-beta/apps/qos/mininet-add-queues.py

En la *Figura 12* se puede ver el estado final de la red creada:

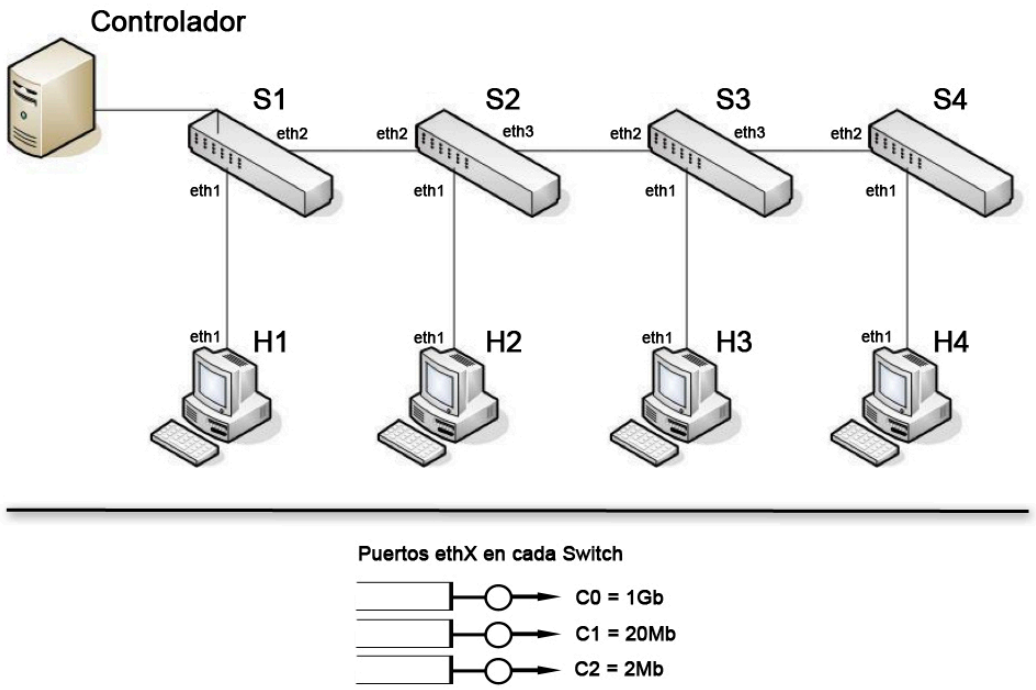

Figura 12: *Red definitiva*

3.1.2. Preparación del controlador Floodlight

En este proyecto se usará una versión de Floodlight diferente a la ofrecida en la página web oficial. Esta versión incluye dos herramientas que facilitarán el desarrollo de este trabajo: el módulo de QoS y el de Firewall.

Para obtener una copia desde el terminal se navegará hasta el directorio donde se quiera trabajar, y se ejecutará:

$git clone https://github.com/wallnerryan/floodlight-qos-beta.git

$cd floodlight-qos-beta

$ant;

El controlador ya estaría listo para ser ejecutado, pero con el objetivo de poder modificar el código en un entorno de desarrollo más preparado, se abrirá el proyecto con el programa Eclipse Luna. Una vez abierto se procederá a la primera compilación y ejecución desde Eclipse. Para ello, desde el menú de opciones Run Configurations se seleccionará como Main Class la clase "net.floodlightcontroller.core.Main".

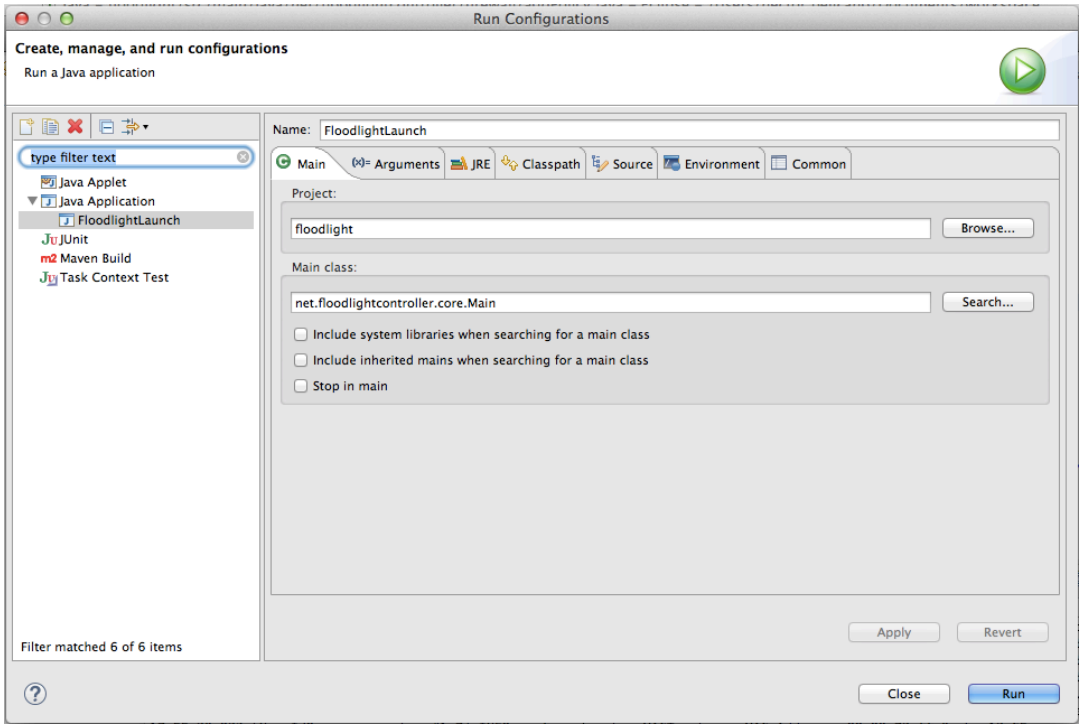

Figura 13: *Configuración de la clase principal ejecutable*

Con la clase Main seleccionada ya se puede ejecutar el controlador que tras unos segundos iniciará el servicio. Se puede acceder al interfaz gráfico introduciendo la url 'http://localhost:8080/ui/index.html' en un navegador.

El interfaz gráfico muestra un resumen de los componentes de la red, la topología de red, los detalles de cada Switch y host, y finalmente las herramientas QoS y Firewall que se incluyen por defecto en este proyecto.

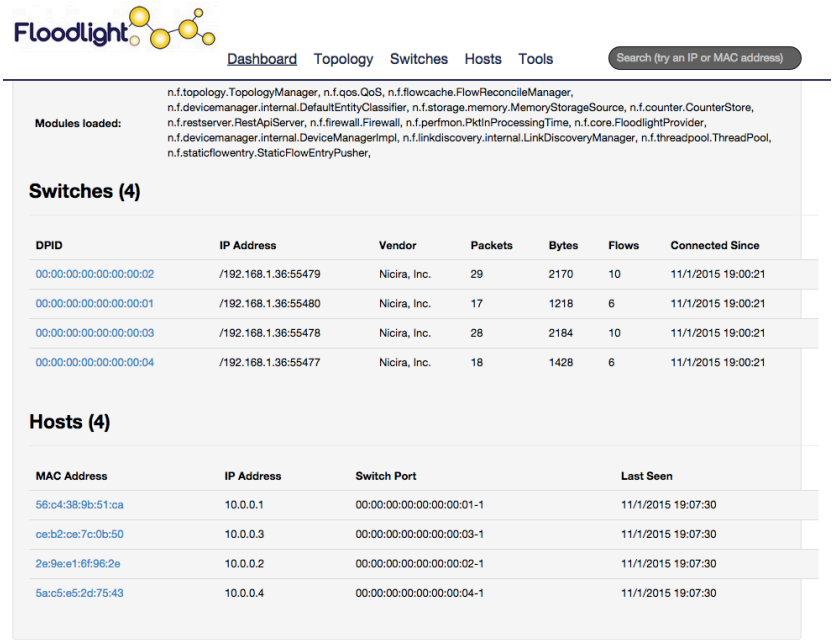

Figura 14: *Interfaz gráfico de Floodligth*

3.2. Modificación del código

Con el entorno de trabajo listo y después de hacer algunas pruebas combinando Mininet y Floodlight, se plantearon diferentes opciones para poder ofrecer QoS dinámica que cambiara sus parámetros en función del tipo de tráfico o del host origen/destino. Para ello era necesario monitorizar los paquetes que pasaran por la red. La solución a este problema se encontró en el módulo Firewall incluido en esta versión de Floodlight.

Dada las limitaciones del Firewall incluido en Floodlight, se decidió que la mejor opción era monitorizar los paquetes en función de las direcciones IP origen y destino. Una vez el Firewall encontrara coincidencias entre las reglas establecidas y los paquetes analizados, se activaría el módulo de QoS automáticamente y se establecerían las políticas predefinidas.

Tanto el Firewall como el módulo de Qos pueden activarse o desactivarse usando el interfaz gráfico de Floodlight, aunque este interfaz no permite gestionar ni las reglas del Firewall ni las políticas de QoS. Con el propósito de automatizar la gestión de ambos módulos, se lanzarán las peticiones usando las aplicaciones REST de Floodlight.

Los clases java modificadas del código principal de Floodlight fueron la clase Firewall.java y la QoSPoliciesResources.java.

En la primera clase se añadieron llamadas a unas clases creadas específicamente para este proyecto, y en la segunda se añadió una funcionalidad para borrar políticas de QoS de manera genérica.

Estas nuevas clases creadas fueron: enableQos.java , disableQos.java y addPolicy.java.

El funcionamiento del sistema completo es el siguiente:

- El usuario define previamente las políticas de calidad en función de un tráfico específico en la red. Estas políticas se han añadido al código de addPolicy.java, aunque también se pueden añadir en cualquier momento desde un terminal.

- También se definen las reglas del Firewall, esta vez, lanzando peticiones desde el terminal.

- En la clase Firewall se analizarán los paquetes transmitidos y se compararán con las reglas existentes. En el caso de haber coincidencia, una petición para iniciar el módulo de calidad será enviada. Seguidamente manda el número de prioridad de la regla a la clase addPolicy.java para que esta añada las reglas definidas previamente al módulo de QoS.

- La clase Firewall también manda una petición para desactivar el módulo de QoS en el caso de que no hubieran coincidencias en un tiempo predefinido.

- En el caso de que dos configuraciones de QoS diferentes fueran solicitadas, siempre se mantendría la de mayor prioridad.

El código de las clases modificadas y creadas se puede consultar en los anexos.

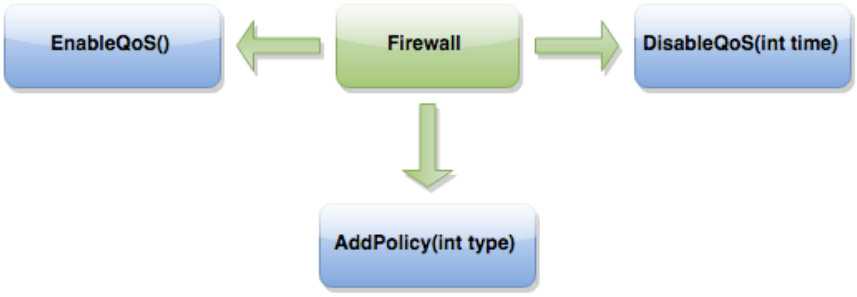

Figura 15: *Dependencia entre las clases creadas y la clase Firewall*

4. Resultados

Para verificar el correcto funcionamiento del programa se planteó una situación donde se pudiera observar un claro cambio en el comportamiento de una red en función de los tráficos existentes.

Utilizando la red definida previamente, se planteó el caso de que la velocidad del tráfico enviado desde el host H2 al host H1 se viera reducido en función de los hosts H3 y H4. Los casos planteados eran los siguientes:

- En el caso de no existir tráfico proveniente de H3 o H4 hacia H1, H2 podría enviar por cualquier canal de los switches S1 y S2, escogiendo de esta manera el canal C0 (más rápido).

- En el caso que H3 enviara datos a H1, todo el tráfico de H2 hacia H1 sería desviado a al canal C1 en los switches S1 y S2.

- En el caso que H4 enviara datos a H1, todo el tráfico de H2 hacia H1 sería desviado a el canal C2 (más lento) en los switches S1 y S2. En este caso el tráfico enviado por H3 hacia H1 también sería desviado al canal C1 de los switches S1, S2 y S3.

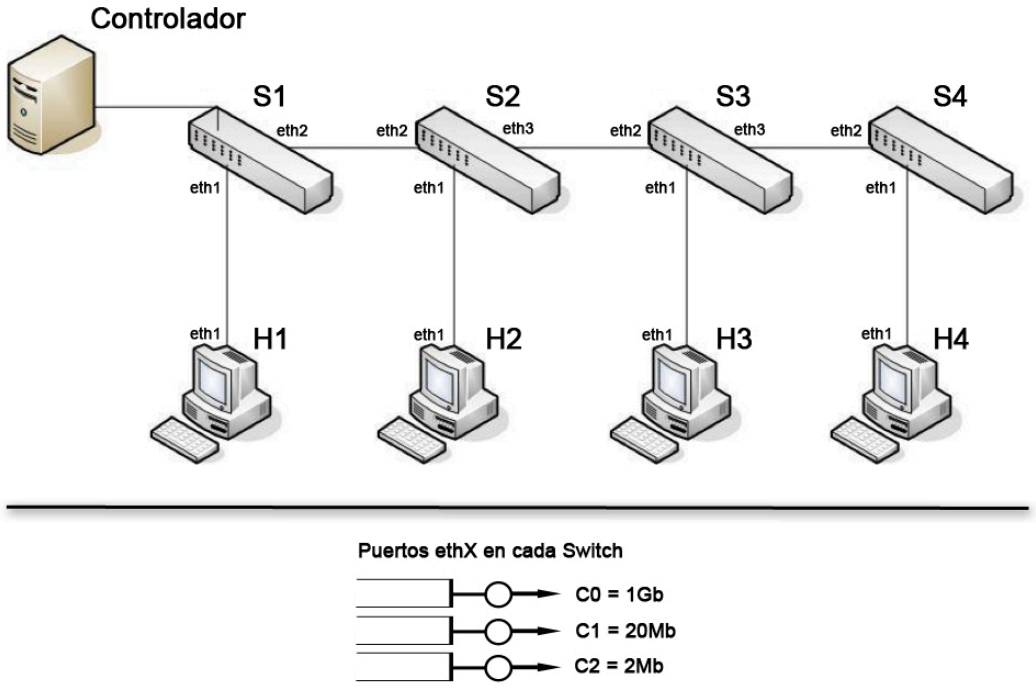

Figura 16: *Red ejemplo*

Con la red creada y controlada por Floodlight desde un terminal de la máquina host se inició el firewall y se le añadieron la reglas adecuadas:

```
$ curl  http://localhost:8080/wm/firewall/module/enable/json

$ curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:01","priority":"1000"}' http://localhost:8080/wm/firewall/rules/json

$ curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:02","priority":"1001"}' http://localhost:8080/wm/firewall/rules/json

$ curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:03","priority":"1002"}' http://localhost:8080/wm/firewall/rules/json

$ curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:04","priority":"1003"}' http://localhost:8080/wm/firewall/rules/json

$ curl -X POST -d '{"src-ip": "10.0.0.3","dst-ip": "10.0.0.1","priority":"99"}' http://localhost:8080/wm/firewall/rules/json

$ curl -X POST -d '{"src-ip": "10.0.0.4","dst-ip": "10.0.0.1","priority":"98"}' http://localhost:8080/wm/firewall/rules/json
```

El primer comando activa el firewall que, si no se le añade ninguna regla, excluirá todos los tráficos. Los cuatro comandos siguientes permiten el paso de cualquier paquete a través de cada uno de los switches. Los dos últimos son los que detectan los paquetes que van desde los hosts H3 y H4 hacia H1. Las prioridades con un número menor tiene preferencia ante las que tienen una número mayor.

En este momento el entorno ya es autosuficiente y gestiona la calidad automáticamente.

La prueba que se realizó para verificar el correcto funcionamiento, fue medir el caudal disponible de H2 para enviar paquetes a H1 usando la herramienta Iperf. Para realizarla, se inicia en el terminal de H1 un servidor Iperf:

h1$ iperf -s

En un primer momento y, con el módulo de QoS todavía desactivado (dado que no habían existido transmisiones entre hosts), se procedió a medir desde H2 el caudal disponible:

h2$ iperf -c 10.0.0.1

Como respuesta, se obtuvo un caudal disponible de unos 453 Mbits/sec. La misma prueba se realizó desde el host H3 obteniendo un caudal de unos 393 Mbits/s.

Para forzar el segundo caso definido se lanzó un ping desde H3 a H1, activando así, el módulo de QoS y añadiendo las políticas pertinentes.

Se realizaron las mismas pruebas que en el primer caso, obteniendo como caudal disponible 18.7 Mbits/sec para H2 y 411 Mbits/s para H3. Se puede ver en este caso como el tráfico de H2 fue desviado a una cola de menor capacidad y que H3 continuó enviando por la cola más rápida.

Finalmente, para verificar el tercer caso se lanzó un ping desde H4 a H1, y se procedió a realizar las medidas de caudal de H2 y H3. En este caso el caudal de H2 fue de 1,90 Mbits/sec y el H3 fue de 19,2Mbits/sec.

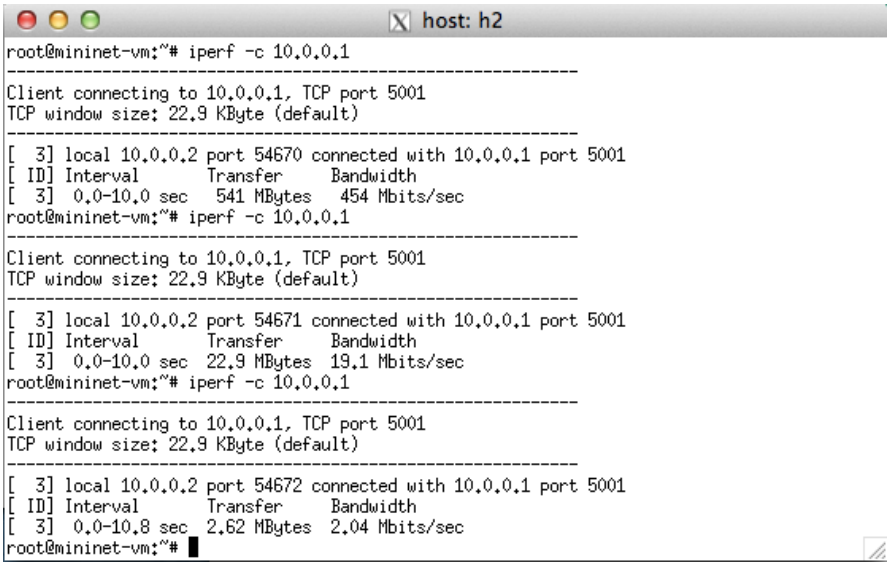

Figura 17: *Pruebas desde Host H2*

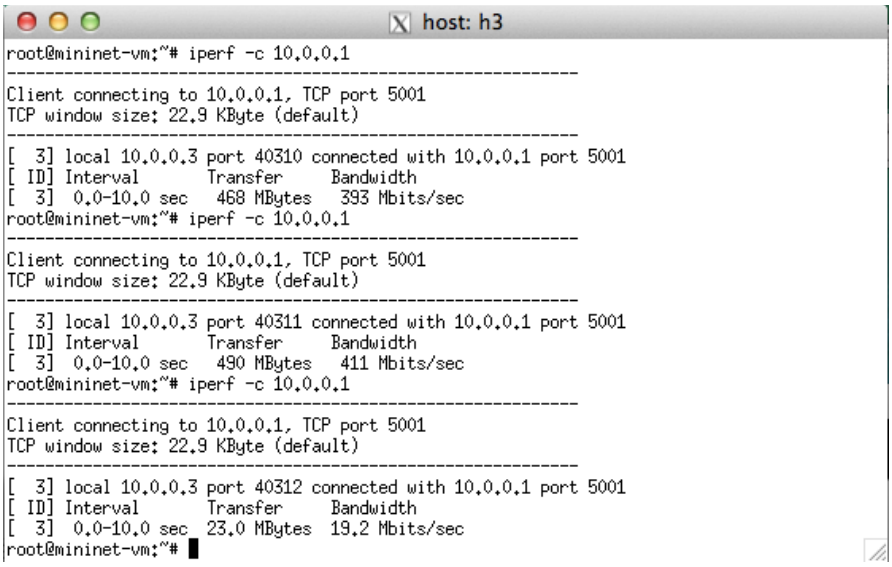

Figura 18: *Pruebas desde Host H3*

En las *Figuras 17 y 18* se pueden ver los iperfs realizados desde los hosts H2 y H3 en cada una de las situaciones. Para hacerlo más visual se han trasladado estos resultado a una gráfica donde se puede comparar el caudal máximo de cada Host en función de las transmisiones de los otros.

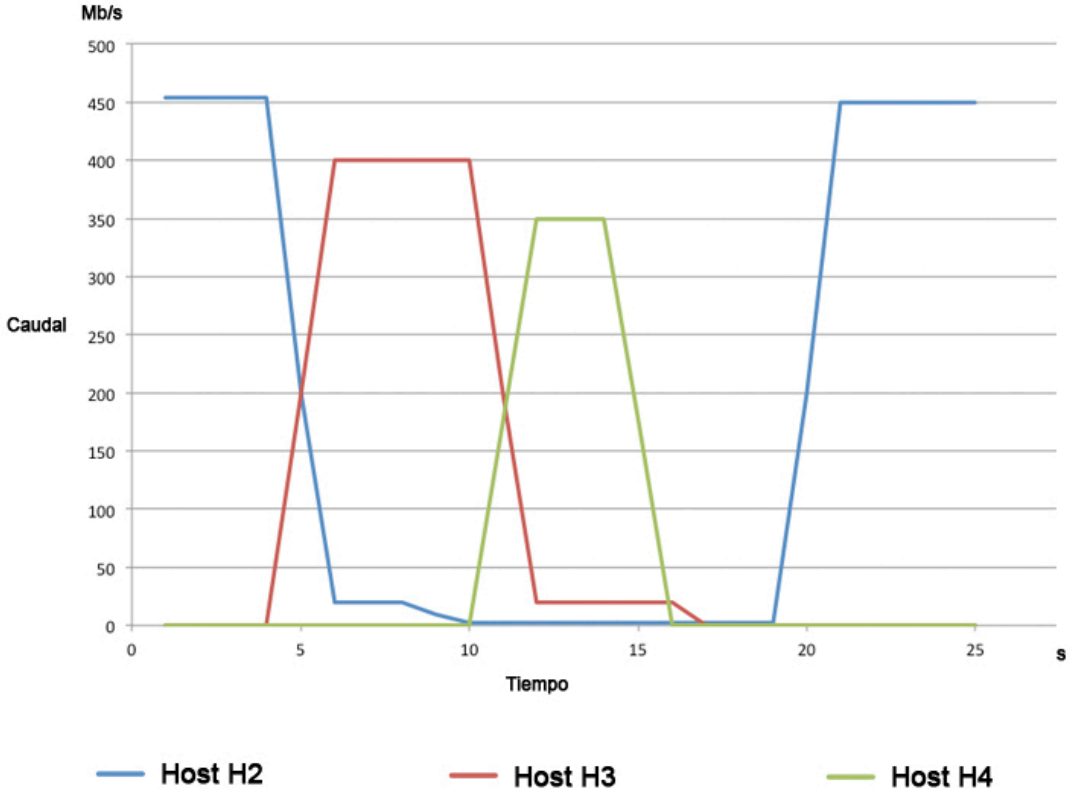

Figura 19: *Gráfica temporal de caudales*

Se puede observar como la tasa de transferencia del host H2 se ha reducido a 20 Mb (Canal C1) tras la primera transmisión del host H3 y a 2Mb (Canal C2) cuando H4 ha empezado a transmitir. Tras un timeout donde ni H3 ni H4 han transmitido paquetes, las políticas de QoS se borran y el host H2 puede volver a transmitir por el canal C0.

Un comportamiento similar se puede observar en la tasa de transferencia del host H3 que pasa de transmitir por el canal C0 a transmitir por el canal C1.

5. Presupuesto

Este proyecto ha sido elaborado al 100% con herramientas Opensource, por lo tanto, el coste se reduce al equipo sobre el que se ha trabajado y el salario del Ingeniero.

Se pueden desglosar los costes como:

- Sueldo Ingeniero Junior a media jornada durante 5 meses: 700 * 5 = 3500 €
- Ordenador portátil: 1100€

Ambos conceptos harían que el presupuesto total de este proyecto sea de 4600€.

6. Conclusiones y futuro desarrollo

Al inicio de este trabajo se buscaron métodos que nos permitieran ofrecer calidad de servicio dinámica sobre redes SDN. Después de barajar diversas opciones, la más adecuada y más conveniente usar fue la combinación del controlador SDN Floodlight y del simulador de redes Mininet, ya que Floodlight incorporaba un módulo de QoS y además funcionaba a la perfección con Mininet, el simulador SDN por excelencia.

Combinando los módulos de Firewall y de QoS incorporados en Floodlight, y incorporando las clases java necesarias (descritas en el apartado 2.1 y 2.2), se ha conseguido monitorizar los paquetes transmitidos en toda la red y, activar en función de estos, las políticas de calidad de servicio predefinidas para cada situación.

En el ejemplo realizado en este trabajo, donde 4 hosts se comunican entre ellos mediante 4 switches interconectados en una tipología del tipo lineal, se puede observar como el caudal disponible para un host se ve modificado dinámicamente en función de los paquetes enviados por otros hosts en la misma red.

En estos ejemplos se han utilizado unas políticas de calidad muy sencillas, como ha sido el desvío de diferentes flujos de datos a ciertos canales de transmisión en función de otros flujos. Estos parámetros podrían haber sido mucho más complejos si el módulo de QoS hubiera sido más completo, cosa que ampliaría en gran medida las utilidad de este programa.

Por lo tanto, con los resultados obtenidos se puede confirmar que se puede ofrecer calidad de servicio dinámica en función del tráfico y, que por lo tanto, el objetivo principal de este proyecto se ha alcanzado.

Del mismo modo que las políticas de calidad, en el ejemplo ofrecido se distinguen los tráficos en función del origen y destino, un siguiente paso en el desarrollo de la aplicación sería el de poder distinguir los paquetes en función del servicio o aplicación usados por el usuario. Para lograr esto, sería necesario desarrollar el módulo firewall de Floodlight y dotarlo de la capacidad necesaria para poder identificar otros parámetros de los paquetes analizados.

Por otro lado, otro factor que podría resultar interesante para este programa sería el de contar con un interfaz gráfico de control que permitiera a cualquier usuario configurar ciertos tipos de tráficos o aplicaciones con mayor preferencia a otras. De esta manera se podría priorizar los datos de una videollamada para que sufriera el menos retardo posible o en otro caso priorizar la descarga de un fichero a la máxima velocidad disponible.

Como comentario personal, decir que tras descubrir las redes SDN y como consumidor de servicios donde el retardo de la señal es fundamental, me veo en condiciones de asegurar que las redes SDN son el futuro y que la investigación sobre estas ha de continuar. La tecnología SDN demuestra aprovechar mucho mejor los recursos que las tecnologías actuales y multitud de empresas emergentes empiezan a confiar cada día más en esta nueva tecnología. Ahora bien, no se espera que las redes acaben substituyendo a las redes actuales, más bien convivirán con ellas aprovechando lo mejor de cada sistema.

Ahora solo nos queda esperar a ver que les depara el futuro a las redes SDN.

Bibliografía

[1] The battle of SDN vs. NFV [Online] Available:

http://www.moorinsightsstrategy.com/the-battle-of-sdn-vs-nfv/

[2] S. Palm, “UPnP QoS Architecture Version 2.0” [Online] Available:

http://upnp.org/specs/qos/UPnP-qos-Architecture-v2.pdf

[3] A. Campbell, C. Aurrecoechea, L. Hauw, “A Review of QoS Architectures” [Online] Available:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.8728&rep=rep1&type=pdf

[4] Ministerio de Industria, Energía y Turismo [Online] Available:

http://www.minetur.gob.es/telecomunicaciones/eses/servicios/calidadservicio/paginas/calidad.aspx

[5] Calidad de Servicio (QoS) [Online] Available:

http://cic.puj.edu.co/wiki/lib/exe/fetch.php?media=materias:daysenr:daysenr_-_calidad_de_servicio_qos_.pdf

[6] Floodlight [Online] Available:

http://docs.projectfloodlight.org/display/floodlightcontroller/The+Controller

[7] Representational State Transfer [Online] Available:

http://es.wikipedia.org/wiki/Representational_State_Transfer

Apéndices

Código modificado de la clase Firewall.java:

```
protected RuleWildcardsPair matchWithRule(IOFSwitch sw, OFPacketIn pi,
        FloodlightContext cntx) {
    FirewallRule matched_rule = null;
    Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
            IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
    WildcardsPair wildcards = new WildcardsPair();


    synchronized (rules) {
        Iterator<FirewallRule> iter = this.rules.iterator();
        FirewallRule rule = null;
        // iterate through list to find a matching firewall rule
        while (iter.hasNext()) {
            // get next rule from list
            rule = iter.next();

            // check if rule matches
            if (rule.matchesFlow(sw.getId(), pi.getInPort(), eth, wildcards) == true) {
                matched_rule = rule;
                //aquí el codigo añadido

                System.out.println(rule.priority);
                    enableQos en = new enableQos();
                    en.start();
                    disableQos dis = new disableQos(50000);
                    dis.start();
                    addPolicy pol = new addPolicy(rule.priority);
                    pol.start();

                //hasta aquí


                break;
            }
        }
    }

    // make a pair of rule and wildcards, then return it
    RuleWildcardsPair ret = new RuleWildcardsPair();
    ret.rule = matched_rule;
    if (matched_rule == null || matched_rule.action == FirewallRule.FirewallAction.DENY) {
        ret.wildcards = wildcards.drop;
    } else {
        ret.wildcards = wildcards.allow;
    }
    return ret;
}
```

Clase creada enableQos.java:

```
package net.floodlightcontroller.firewall;


import java.io.IOException;


public class enableQos extends Thread {
	int secs;
	String command1 ;
	public enableQos(){

	}

	public void run(){


		try {
			Runtime.getRuntime().exec("curl
http://localhost:8080/wm/qos/tool/enable/json");
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}


	}
}
```

Clase creada disableQos.java:

```
package net.floodlightcontroller.firewall;


import java.io.IOException;


public class disableQos extends Thread {
	int secs;
	String command1 ;
	public disableQos(int secs){
		this.secs=secs;

	}

	public void run(){


		try {
	   //thread to sleep for the specified number of milliseconds
	   Thread.sleep(secs);
	} catch ( java.lang.InterruptedException ie) {
	   System.out.println(ie);
	}
```

```
		try {
			for(int i = 0; i < 10; i++ ){
			Runtime.getRuntime().exec("curl -X DELETE -H \"Content-Type: application/json\" -d {\"policy-id\":0} http://localhost:8080/wm/qos/policy/json ");
			}
			} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
		try {
			Runtime.getRuntime().exec("curl http://localhost:8080/wm/qos/tool/disable/json");
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}


	}


}
```

Clase creada addPolicy.java

```
package net.floodlightcontroller.firewall;


import java.io.IOException;


public class addPolicy extends Thread {
	int secs;
	int type;
	String command1 ;
	public addPolicy(int type){
		this.secs=20000;
		this.type=type;

	}

	public void run(){


	if(type==99){
	try {
	for(int i = 0; i < 10; i++ ){
	Runtime.getRuntime().exec("curl -X DELETE -H \"Content-Type: application/json\" -d {\"policy-id\":0} http://localhost:8080/wm/qos/policy/json ");
		}
	Runtime.getRuntime().exec("curl -X POST -H \"Content-Type: application/json\" -d {\"name\":\"host2acola2enS2\",\"protocol\":\"6\",\"eth-type\":\"0x0800\",\"ip-src\":\"10.0.0.2\",\"ip-dst\":\"10.0.0.1\",\"sw\":\"00:00:00:00:00:00:00:02\",\"queue\":\"1\",\"enqueue-port\":\"2\"} http://localhost:8080/wm/qos/policy/json ");
	Runtime.getRuntime().exec("curl -X POST -H \"Content-Type: application/json\" -d {\"name\":\"host2acola2enS1\",\"protocol\":\"6\",\"eth-type\":\"0x0800\",\"ip-src\":\"10.0.0.2\",\"ip-
```

```
dst\":\"10.0.0.1\",\"sw\":\"00:00:00:00:00:00:00:01\",\"queue\":\"1\",\"enqueue-port\":\"1\"}
http://localhost:8080/wm/qos/policy/json ");
		} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
		}

		}
if(type==98){
		try {

		for(int i = 0; i < 10; i++ ){
		Runtime.getRuntime().exec("curl -X DELETE -H \"Content-Type: application/json\" -d
{\"policy-id\":0} http://localhost:8080/wm/qos/policy/json ");
		}
		Runtime.getRuntime().exec("curl -X POST -H \"Content-Type: application/json\" -d
{\"name\":\"host2acola3enS2\",\"protocol\":\"6\",\"eth-type\":\"0x0800\",\"ingress-port\":\"1\",\"ip-
src\":\"10.0.0.2\",\"sw\":\"00:00:00:00:00:00:00:02\",\"queue\":\"2\",\"enqueue-port\":\"2\"}
http://localhost:8080/wm/qos/policy/json ");
		Runtime.getRuntime().exec("curl -X POST -H \"Content-Type: application/json\" -d
{\"name\":\"host2acola3enS1\",\"protocol\":\"6\",\"eth-type\":\"0x0800\",\"ingress-port\":\"2\",\"ip-
src\":\"10.0.0.2\",\"sw\":\"00:00:00:00:00:00:00:01\",\"queue\":\"2\",\"enqueue-port\":\"1\"}
http://localhost:8080/wm/qos/policy/json ");
		Runtime.getRuntime().exec("curl -X POST -H \"Content-Type: application/json\" -d
{\"name\":\"host3acola2enS3\",\"protocol\":\"6\",\"eth-type\":\"0x0800\",\"ingress-port\":\"1\",\"ip-
src\":\"10.0.0.3\",\"sw\":\"00:00:00:00:00:00:00:03\",\"queue\":\"1\",\"enqueue-port\":\"2\"}
http://localhost:8080/wm/qos/policy/json ");
		Runtime.getRuntime().exec("curl -X POST -H \"Content-Type: application/json\" -d
{\"name\":\"host3acola2enS2\",\"protocol\":\"6\",\"eth-type\":\"0x0800\",\"ingress-port\":\"3\",\"ip-
src\":\"10.0.0.3\",\"sw\":\"00:00:00:00:00:00:00:02\",\"queue\":\"1\",\"enqueue-port\":\"2\"}
http://localhost:8080/wm/qos/policy/json ");
		Runtime.getRuntime().exec("curl -X POST -H \"Content-Type: application/json\" -d
{\"name\":\"host2acola2enS1\",\"protocol\":\"6\",\"eth-type\":\"0x0800\",\"ingress-port\":\"2\",\"ip-
src\":\"10.0.0.3\",\"sw\":\"00:00:00:00:00:00:00:01\",\"queue\":\"1\",\"enqueue-port\":\"1\"}
http://localhost:8080/wm/qos/policy/json ");
		}catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
		}

		}

}

}
```

Glosario

- Caudal: Cantidad de ocupación en un ancho de banda.

- Ethernet: Estándar de redes de área local para computadores con acceso al medio por detección de la onda portadora y con detección de colisiones.

- Frame Relay: Técnica de comunicación mediante retransmisión de tramas para redes de circuito virtual.

- GET: Método de petición de datos. La longitud de la petición GET está limitada por el espacio libre en los buffers de entrada.

- Github: Servicio web para la gestión de código y programas en desarrollo.

- ICANN: Internet Corporation for Assigned Names.

- IEEE: Institute of Electrical and Electronics Engineers.

- NAT: Mecanismo utilizado por routers IP para intercambiar paquetes entre dos redes que asignan mutuamente direcciones incompatibles.

- OpenFlow: Lenguaje usado entre switches SDN para comunicarse.

- Open Source: Expresión con la que se conoce al software distribuido y desarrollado libremente.

- POST: Método de petición de datos. No sufre limitaciones de espacio y puede enviar mucha más información al servidor que una petición GET.

- Prompt: Carácter o conjunto de caracteres que se muestran en una línea de comandos para indicar que está a la espera de órdenes.

- Python: Lenguaje de programación interpretado que soporta orientación a objetos.

- REST: Técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.

- Retardo: Tiempo que tarda una señal en recorrer una ruta marcada.

- SSH: Protocolo que permite controlar equipos remotamente.

- Tasa de transferencia: Número de bits que se transmiten por unidad de tiempo a través de un sistema de transmisión digital o entre dos dispositivos digitales.

- VoIP: Grupo de recursos que hacen posible que la señal de voz viaje a través de Internet empleando un protocolo IP.

- Wireshark: Analizador de protocolos.